Progetto di Laboratorio Java
Programmazione ed Analisi dei Dati

Paolo Labruna
Matricola 532622
Anno Accademico 2020/2021

Le specifiche iniziali.

Il progetto ha come obbiettivo quello di creare un grafico delle frequenze di frequenze delle parole contenute nei file di testo contenuti nei folder che vanno da 1 a 9 del progetto Guntenberg (nella versione contenuta nell'iso ftp://mirrors.pglaf.org/mirrors/gutenberg-iso/pgdvd042010.iso) . Nei folder i testi sono memorizzati in file compressi che vanno quindi scompattati. I file di testo da prendere in esame sono solamente quelli in inglese, nel caso vi fossero più file con lo stesso nome se ne deve considerare solamente uno. I singoli testi devono essere aperti una prima volta per controllarne la lingua e acquisirne l'encoding, successivamente si devono riaprire utilizzando l'encoding letto e procedere con l'elaborazione per calcolare la frequenza delle parole e la frequenza delle frequenze. All'interno dei testi solo le parole comprese tra lo start e l'end (determinati rispettivamente dalle sequenze di caratteri "*** START" e "*** END") vengono conteggiate.

Le scelte implementative.

Il grafico finale è stato ottenuto facendo un copia incolla dei dati prodotti dal programma WordFreq.java in un foglio elettronico excel sul quale è stato inserito un grafico XY con entrambi i formati degli assi in scala logaritmica a base 10.
Il programma prende come unico argomento facoltativo (con default ./pgdvd042010) il percorso con la cartella contenente i folder che saranno scanditi ricorsivamente e restituisce in output due file di testo. Il primo, result.txt, oltre a contenere i dati che ci consentono di creare il grafico (le coppie parole/frequenze) contiene anche il numero di file univoci di testo in inglese processati, il numero dei file zip che sono stati decompressi, il numero totale di token e il numero di parole univoche. Nel secondo file (fileList.txt) è contenuta la lista dei nomi di tutti i file di testo che sono stati considerati. Il programma scandisce ricorsivamente la directory data in input, processa i file txt che incontra e decomprime gli zip. Lo scompattamento dei file avviene in memoria, i dati non vengono scritti nel file system. La frequenza dei token viene memorizzata in una HashMap mentre per la lista dei file di testo che vengono presi in esame viene utilizzata una HashSet. Una volta conclusa la scansione le frequenze delle frequenze vengono scritte in una TreeMap attraverso la rilettura della HasMap. L'esecuzione del programma ha messo in evidenza nei file di testo alcuni encoding non direttamente comprensibili da java (in alcuni casi si tratta di errori di digitazione), si è quindi cercato di ottenere una corretta interpretazione dell'encoding.

Esempi di funzionamento

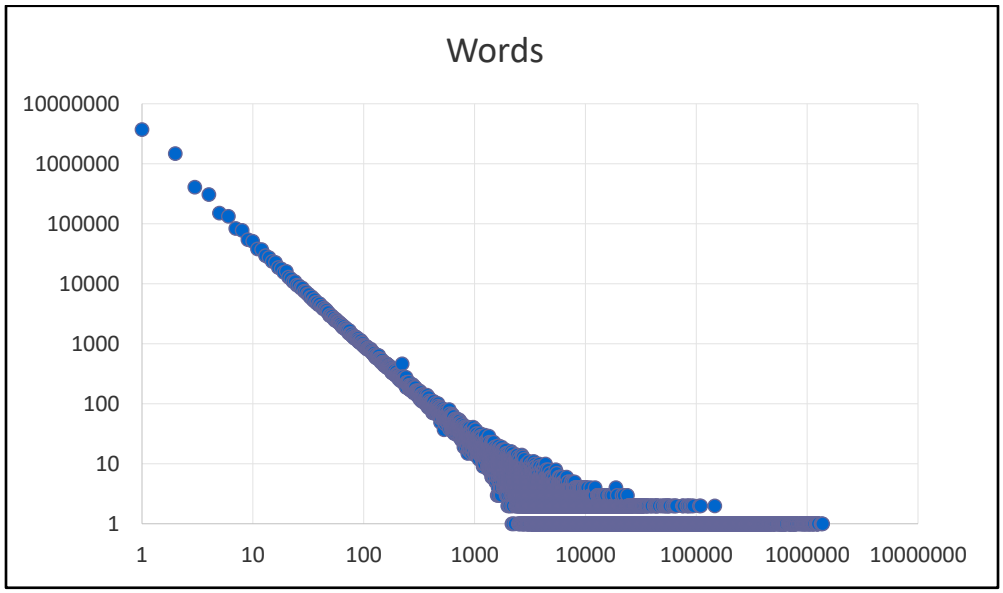
Il programma eseguito utilizzando come parametro la cartella contenente i 9 folder del progetto Gutenberg produce il file result.txt con il seguente contenuto (vengono riportate solo le prime 10 coppie parole/frequenza):

Unique Txt files (English):	21862
Zip files:	28530
Tokens:	1389532368
Words:	7145137

Words	Freq
3719789	1
1483119	2
408907	3
307147	4
150142	5
133001	6
83334	7
77611	8
54661	9
51523	10

……

Utilizzando le coppie parole/frequenza contenuti in questo file è possibile costruire, tramite excel, il grafico (in scala doppio logaritmica a base 10) richiesto dalle specifiche:



Il listato del programma.

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.StringTokenizer;
import java.util.TreeMap;
import java.util.concurrent.atomic.AtomicLong;
import java.util.zip.ZipEntry;
import java.util.zip.ZipFile;
import java.util.zip.ZipInputStream;

public class WordFreq {

	final static String DELIM = " ,.;?!\"\'";
	final static String LANG = "English";
	final static String DEFAULT_ENCODING = "UTF-8";
	final static String DEFAULT_DIR = "./pgdvd042010";

	public static void main(String[] args) throws IOException {
		String path = DEFAULT_DIR;
		if (args.length > 0 && !args[0].trim().isEmpty())
			path = args[0];
```

```
File f = new File(path);
if (!f.isDirectory()) {
        System.out.println("Folder not found: " + path);
        return;
}

AtomicLong zipCount = new AtomicLong();
long tokensCount = 0;

Map<Object, Integer> h = new HashMap<Object, Integer>(); // token, freq.
Map<Object, Integer> t = new TreeMap<Object, Integer>(); // freq., word
HashSet<String> txtList = new HashSet<String>(); // txt file list

Files.walk(Paths.get(path)).filter(Files::isReadable).filter(Files::isRegularFile)
.filter(file -> (file.toString().toLowerCase().endsWith(".txt")
                || file.toString().toLowerCase().endsWith(".zip")))
.forEach(file -> {
        try {
                ProcessFile(file, h, txtList, zipCount);
        } catch (IOException e) {
                e.printStackTrace();
        }
});

for (Integer v : h.values()) {
        add(t, v);
        tokensCount += v;
}

FileWriter fw = new FileWriter("result.txt");

fw.write("Unique Txt files (" + LANG + "):\t" + txtList.size() + "\n");
fw.write("Zip files:\t" + zipCount.get() + "\n");
fw.write("Tokens:\t" + tokensCount + "\n");
fw.write("Words:\t" + h.size() + "\n\n");

fw.write("Words\t\tFreq\n\n");
for (Object o : t.keySet())
```

```
                fw.write(t.get(o) + "\t\t" + o.toString() + "\n");

        fw.close();

        FileWriter fl = new FileWriter("fileList.txt");
        for (String txt : txtList)
                fl.write(txt + "\n");
        fl.close();

}

private static void ProcessFile(Path f, Map<Object, Integer> h, HashSet<String> fl, AtomicLong zipCount)
                throws IOException {
        if (f.toString().toLowerCase().endsWith(".txt") && !fl.contains(f.getFileName().toString().toLowerCase())) {
                BufferedReader in = new BufferedReader(new FileReader(f.toString(), Charset.forName(DEFAULT_ENCODING)));
                String encod = checkTxt(in, LANG);
                if (!encod.isEmpty()) {
                        BufferedReader inBuff = new BufferedReader(new FileReader(f.toString(), Charset.forName(encod)));
                        read(inBuff, h);
                        fl.add(f.getFileName().toString().toLowerCase());
                }
        } else if (f.toString().toLowerCase().endsWith(".zip")) {
                Unzip(f.toString(), h, fl);
                zipCount.incrementAndGet();
        }
}

private static void read(BufferedReader in, Map<Object, Integer> h) {
        try {
                boolean startFound = false;
                String line = in.readLine();
                while (line != null) {
                        if (line.startsWith("***END") || line.startsWith("*** END"))
                                break;

                        if (startFound) {
                                StringTokenizer st = new StringTokenizer(line, DELIM);
                                while (st.hasMoreTokens())
```

```
                                        add(h, st.nextToken());
                                } else if (line.startsWith("***START") || line.startsWith("*** START"))
                                        startFound = true;

                                line = in.readLine();
                        }

                        in.close();

                } catch (IOException e) {
                        e.printStackTrace();
                }
        }

        private static void add(Map<Object, Integer> m, Object v) {
                Integer o = m.putIfAbsent(v, 1);
                if (o != null)
                        m.put(v, o + 1);
        }

        private static void Unzip(String zipFile, Map<Object, Integer> h, HashSet<String> fl) throws IOException {
                ZipInputStream zis = new ZipInputStream(new FileInputStream(zipFile));
                ZipEntry zipEntry = zis.getNextEntry();
                while (zipEntry != null) {
                        String st = new File(zipEntry.getName()).getName().toLowerCase();

                        if (st.endsWith(".txt") && !fl.contains(st)) {
                                BufferedReader in = new BufferedReader(new InputStreamReader(zis, DEFAULT_ENCODING));
                                String encod = checkTxt(in, LANG);
                                if (!encod.isEmpty()) {
                                        ZipFile zip = new ZipFile(zipFile);
                                        ZipEntry entry = zip.getEntry(zipEntry.getName());

                                        BufferedReader inBuff = new BufferedReader(new InputStreamReader(zip.getInputStream(entry),
encod));
                                        read(inBuff, h);
                                        fl.add(st);
                                        zip.close();
                                }
```

```
                }
                zipEntry = zis.getNextEntry();
        }
        zis.closeEntry();
        zis.close();
}

private static String checkTxt(BufferedReader in, String Lang) throws IOException {
        String enc = "";
        String lang = "";
        String line = in.readLine();

        while (line != null) {
                if (line.startsWith("Language: ")) {
                        lang = line.substring(10);
                        if (lang.compareToIgnoreCase(Lang) != 0)
                                return "";
                        else if (enc != "")
                                return enc;

                } else if (line.startsWith("Character set encoding: ")) {
                        String str = line.substring(24).trim();
                        byte[] ba = str.getBytes(StandardCharsets.US_ASCII);
                        enc = new String(ba, StandardCharsets.US_ASCII);

                        if ((enc.compareToIgnoreCase("ISO Latin-1") == 0) || (enc.compareToIgnoreCase("ISO 8859-1") == 0))
                                enc = "ISO-8859-1";
                        else if (enc.compareToIgnoreCase("ISO-646-US (US-ASCII)") == 0)
                                enc = "US-ASCII";
                        else if (enc.compareToIgnoreCase("MP3") == 0)
                                enc = "US-ASCII";
                        else if (enc.compareToIgnoreCase("Unicode UTF-8") == 0)
                                enc = "UTF-8";
                        else if (enc.compareToIgnoreCase("IDO-8859-1") == 0) // scritto male
                                enc = "ISO-8859-1";
                        else if (enc.compareToIgnoreCase("ISO-8858-1") == 0) // non esiste ISO-8858-1 forse il ISO 8859-1 ?
                                enc = "ISO-8859-1";
```

```
                        try {
                                Charset inputCharset = Charset.forName(enc);
                        } catch (Exception e) {
                                System.out.println("ignored encoding " + enc + " assigned to " + DEFAULT_ENCODING);
                                enc = DEFAULT_ENCODING;
                                if (lang != "")
                                        return enc;
                        }

                        // US-ASCII, MIDI, Lilypond, MP3 and TeX

                        if (lang != "")
                                return enc;
                }

                line = in.readLine();
        }

        return "";
    }
}
```